Mutex Overview Solutions

Copyright © James Raynard 2021

Mutex

- Explain how using a mutex allows multiple threads to access shared data safely
  - The threads are written in such a way that they can only access the shared data after acquiring a lock on the mutex
  - If the mutex is locked, this is because another thread is accessing the shared data. The thread must wait until the that thread unlocks the mutex before it can access the shared data
  - When a thread has a lock on the mutex, no other thread will be able to acquire a lock. The thread can access the shared data without being interrupted

Copyright © James Raynard 2021

Data Race Example with std::mutex

- Rewrite your program from the previous lecture so that it uses a mutex to avoid the data race
- Run your program several times to verify that the output is not interleaved

Copyright © James Raynard 2021

Drawbacks of std::mutex

- What happens if a thread calls lock() but does not call unlock()?
  - The lock on the mutex will not be released, even after the thread has completed its execution
  - Any other threads that are waiting for a lock on the mutex will be blocked indefinitely

Copyright © James Raynard 2021

Mutex wrappers

- Which two classes does C++ provide to manage mutexes?
  - std::lock_guard
  - std::unique_lock
- How do these classes avoid the drawbacks of using std::mutex directly?
  - The mutex is unlocked in the class's destructor
  - This ensures that the mutex is always unlocked
- Rewrite your program twice, to use each of these classes to avoid a data race

Copyright © James Raynard 2021